Department: Software Engineering
Editors: Jeffrey C. Carver, carver@cs.ua.edu and Karla Morris, knmorri@sandia.gov.

# Continuous Integration Challenges for HPC

**Todd Gamblin**
Lawrence Livermore National Laboratory

**Daniel S. Katz**
NCSA & CS & ECE & iSchool, University of Illinois at Urbana-Champaign

***Abstract*—Continuous integration (CI) is ubiquitous in modern software development. Major code hosting services provide free CI automation on common platforms. There are great benefits to using it: CI catches bugs in code *before* they are committed. High performance computing (HPC) research depends heavily on software, but HPC machines are not "common" platforms. A number of challenges stand in the way of CI for HPC environments. Without CI, it is much harder to keep HPC projects bug-free. The research community suffers as a result.**
**We examine the challenges to HPC CI, including hardware diversity, security, isolation, administrative policies, and non-standard authentication, environments, and job submission mechanisms. We suggest a number of solutions. Most will require significant changes at HPC centers, but they could greatly improve HPC software quality and developer experience, which will ultimately enable better, faster science.**

**HIGH PERFORMANCE COMPUTING** is a key enabler for developing scientific understanding and knowledge. "High performance" typically refers to computing that requires large-scale resources, e.g., those on the Top500 list of the world's fastest machines [1]. HPC sites range from universities with smaller clusters of commodity machines to large, GPU-accelerated supercomputers at national computing facilities.

HPC systems are not useful without application software. Since around the 1940s, HPC applications have spanned the computational science domains (simulations and modeling in climate, physics, chemistry, engineering, etc.) More recently, the field grew to include applications in data analysis and machine learning. All of these applications rely on other software, from operating systems to libraries (e.g., for communications and math), and still more software is used in the development process: compilation, testing, packaging, and distribution.

Historically, staff at HPC sites developed their own applications, with the vendor of the HPC system providing the operating system, compilers, and math libraries. Export and other data sensitivity concerns limit access to a significant number of HPC applications. Because of these and more general security concerns, HPC sites

 Published by the IEEE Computer Society  

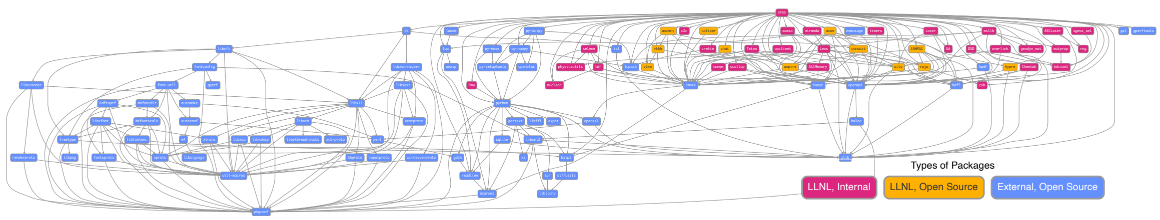


**Figure 1.** Dependencies of the ARES multi-physics code: 72 are external open-source software (OSS), 13 are internal OSS, and 31 are internal proprietary packages.

only grant access to a set of known account holders. However, a large fraction of today's software is developed on social coding platforms like GitHub and GitLab, which allow a community to perform collaborative planning, development, maintenance, and testing. These sites not only provide infrastructure for working on code, but a large number of free cloud CPU cycles for continuous integration (CI). Under the CI model, tests run when developers suggest changes, and the tests ensure that code is correct *before* it is accepted. Developers can thus have high confidence that the code will work correctly.

While continuous integration is standard practice for developers who can use common cloud environments, HPC environments introduce challenges to this practice. Technical, security, and political issues all make it extremely difficult to integrate externally developed open source software with internal applications and machines. Even though many HPC projects are developed in the open, they must run on closed HPC resources, and it is increasingly difficult to ensure that the vast majority of modern *open source* applications will run reliably on HPC systems.

## MODERN SOFTWARE COMPLEXITY

Modern software applications are not monolithic; they integrate packages written by many authors on different project teams, and they rely heavily on publicly-available open-source software. Figure 1 shows the software packages used by ARES, a proprietary multi-physics application used on HPC machines at Lawrence Livermore National Laboratory (LLNL). These packages include core scientific libraries and utility libraries for logging, math, I/O, programming models, performance portability, memory management, and other purposes. There are also a number of build-time dependencies like compilers and testing frameworks.

Even though 30 core components of Ares are LLNL-proprietary, the other 85 packages are open source. Of these, 12 are publicly developed by LLNL on GitHub, and the remaining 71 packages are open-source packages developed by others. This situation is not unusual; *most* modern software leverages and depends on open source. It would be impossible, or at least impractically expensive, for a single organization to reimplement all the capabilities provided by the modern open-source software ecosystem.

The cost of software reuse, however, is integration complexity. On a project developed by one team, that team makes commits to a common repository, which keeps the project consistent. With large, integrated systems, different teams may develop each component, and developers must ensure that versions of *all* components work together. Unfortunately, open-source developers do not have access to HPC resources, and even if they had access, many do not have the time to manually test their packages in HPC-like environments. HPC developers who leverage open source must typically be ready to do (sometimes extensive) porting and integration testing in order to get the open components to work well on closed systems.

## UNIQUENESS OF HPC SYSTEMS

HPC systems are typically designed and built to meet specific local requirements, balancing expected workload characteristics, hardware options (e.g., number and type of CPUs, GPUs, accelerators; internal networking; storage), packaging, cooling, external networking, energy usage, cost, etc. For each system, key components of the local software stack are often bespoke. For example, proprietary MPI implementations like Cray MPI can only run on Cray systems. In this case, Cray MPI's license does not allow inclusion in containers or other software distributions that can run in the cloud. The same is true for math libraries and compilers in the Cray environment. There is typically no standard filesystem organization. Paths to tools, libraries, home, and temporary directories are system-dependent. Authorization and access may be set by site policies, which are often developed locally.

## HPC BUILD & TEST CHALLENGES

Open-source developers are now accustomed to widely available compute cycles for continuous integration. Major code hosting sites (GitHub, GitLab, Bitbucket) as well as third-party paid services that integrate with these sites offer free CI services. Developers can attach workflows to their repositories that run tests concurrently across Linux, macOS, and Windows, and if they need to test in custom environments they can bring their *own* containerized test environments.

### Replicating HPC environments

Due to ubiquitous cloud computing, CI is the norm outside of HPC. It has never been easier to set up automated test environments in *widely-used* software environments, but as discussed above, HPC environments are, by definition, special. For example, it is seldom possible to reliably test *optimized* CPU builds in cloud CI, as the fleet of test systems used by cloud virtual machines (VMs) is often heterogeneous and one cannot request a that a test run on a specific microarchitecture. So far, there is no (free) cloud-based CI for GPUs.

Testing scientific workflow systems is even harder. Each workflow system is essentially a distributed application, and testing a workflow system requires access to the job submission interface. This access can include include authentication and authorization from remote systems, local environments and configurations, batch scheduler parameters, etc. Because resource managers are used to run the CI system itself, it is difficult to vary and test system software and resource managers *within* the CI system. In this case, we need to see how the system is set up in practice, and not completely isolate from it. We also need interfaces and abstractions that allow us to test that the system works across different schedulers and configurations.

Without the ability to replicate the software environment of popular HPC systems, it is very difficult to ensure that open-source software will continue *working* on them.

### Security challenges

HPC machines are large, shared computing systems like clouds, and one obvious way to replicate the HPC environment in cloud CI would be to offer cycles on a local HPC system to run CI jobs for sites like GitHub or GitLab. However, most HPC sites disallow users from running jobs on behalf of external systems.

Consider the open-source CI model, where an unknown user (or at least a user who is unknown to the HPC center) submits a pull request (PR) to a project. The PR triggers jobs that build and test the changed code in cloud environments. Developers and maintainers often want to instead trigger jobs on a set of HPC platforms. While the cloud allows users to provision isolated virtual machines and even isolated virtual networks for CI jobs, most HPC systems lack this level of isolation. Sites implement HPC security at the facility boundary, allowing only trusted users in. Once on the system, all users can access the shared filesystem and can connect to compute nodes over the cluster network. A privilege escalation in this environment could give a user access to other users' files, which may be export-controlled or otherwise sensitive. HPC security teams therefore disallow setting up CI to run arbitrary code, as it opens the site to such attacks.

Running code from *protected* branches, e.g. the maintainer-approved main branch of a popular open-source project, may be allowed, but doing this loses the benefits of testing changes *before* they are integrated into the project. When the PR is still open, contributors are motivated to fix issues that come out in testing because they want their changes to be merged. If fixes are made after the fact, it can be very difficult to keep fast-moving projects working for HPC.

### Administrative and Political challenges

It can be difficult to make progress on solutions to the above problems for a number of administrative and political reasons. First, HPC sites do not typically prioritize build or test cycles, because this is perceived to reduce cycles available for production science runs, which is typically their raison d'être. CI jobs tend to be small and numerous, as opposed to the more traditional larger, longer-running HPC jobs, and queuing policies that support this type of work are not well understood, especially for heavily utilized systems with mostly larger jobs. When asked how many cycles are needed for testing, users reply with very large numbers and the need to test at scale. Facilities are reluctant to provide any one project with a large testing allocation.

The tradeoff between using cycles for testing and saving cycles on production code that may fail is not easy to quantify, but if public CI systems are any benchmark, a large fraction of the benefit of CI can be realized through short-running builds and smoke tests. Cloud CI services impose strict limits on job runtimes and resource usage—typically short numbers of hours and one or two CPUs per job. All but the largest codes can be built and tested for correctness within this footprint, at least at a coarse granularity. Providing separate queues with similar policies on HPC systems would require only a small fraction of overall system CPU hours, and while this approach would not detect bugs that only appear at massive scale, it would still prevent many production cycles from being wasted.

Because HPC sites are very focused on production jobs and production job performance, very little interest has emerged in the HPC community for compute or network virtualization. This is unfortunate, because these technologies would provide the type of resource flexibility needed to run isolated, secure CI jobs. Infiniband, the most popular HPC network, has very limited support for traffic isolation (8 or so isolated channels—not enough for thousands of users), and most HPC systems still run applications on bare metal instead of in VMs. Meanwhile, clouds have developed very lightweight, secure VM solutions (e.g., Amazon Web Service's Nitro hypervisor) with almost no virtualization overhead.

Finally, there is only limited understanding of modern development workflows among HPC center leadership. It is difficult to grasp the extent to which open source has spread throughout the scientific software ecosystem, the rate at which modern software is developed, and the interdependence of packages. The idea that key science applications rely on externally developed software, that helping external software projects test on HPC machines could be *beneficial* to internal projects, and that many *internal* projects are actually hosted and developed externally still needs socializing in order to broaden understanding of the needs of modern software developers.

## POTENTIAL SOLUTIONS

Building and testing software on HPC systems has always been hard, but some solutions have recently begun to emerge.

### Wisdom of the crowds

Systems like Spack [2] and EasyBuild [3] have made building on HPC systems easier by crowd-sourcing institutional build knowledge. These systems include curated repositories of build scripts that aggregate and preserve institutional knowledge of different machine environments and make HPC software easier to build. While the projects themselves require extensive CI, changes are only checked automatically with cloud CI, not on a diverse set of HPC resources. Without immediate, automated testing of contributions, builds still frequently break and tests still frequently fail in these environments.

### Jacamar and secure CI

For *internal*, trusted projects at HPC centers, projects like Jacamar CI [4] solve some of the security problems. They allow users to run CI jobs *as themselves* on HPC machines, preserving the OS-level security boundaries that HPC center require users to adhere to. With Jacamar, one user cannot access and steal another user's data through the CI system. While internal projects *can* pull in trusted versions of external software (e.g. recent releases), integration testing is still difficult. Internal teams cannot easily test changes from PRs, because the changes in a PR cannot be attributed to any trusted HPC center user. Without the ability to test PRs, incompatibilities or bugs can be introduced through dependencies. Either the site must attribute every PR to a known user, which is often not possible, or they must isolate the untrusted code in its own environment.

### Separate resources

HPC sites are considering setting up separate resources for open-source CI. One of the authors has been involved in such an effort at LLNL, to set up an isolated cluster *without* sensitive data, where public CI jobs can run with little risk to the main HPC resources. The challenge with this approach is that it duplicates effort—HPC system administrators themselves are a scarce resource, and maintaining an additional machine in a different network zone requires redundant work. It is also difficult to ensure that the separate machine stays up to date with the main systems.

### Vendor and cloud support

As customers have come to rely on an increasing volume of open source, HPC vendors have shown more interest in ensuring that this software works well on the platforms they offer. More cloud vendors are producing their own HPC offerings, where users can easily set up their own clusters in the cloud to run HPC jobs. These clusters can even use a wide range of resource managers, like SLURM and PBS. However, such environments are not free.

HPC vendors *may* begin to provide free, public cloud CI resources that open-source developers could use to test their software. For large projects like Spack, cycles can be donated in one place, but scaling the approach to support the many smaller, independent HPC development projects that need CI is a much larger effort that requires more cooperation between major HPC vendors and cloud platforms.

### Containerized environments

In lieu of hardware resources, HPC vendors could also begin to provide containerized versions of their software stack for building and testing in CI. Some vendors have begun testing this approach, for example with Cray's Containerized Programming Environment (CPE). Unfortunately, the container is currently only licensed to run on HPC resources, so its versatility for build and test use cases is limited. It cannot be run in the cloud, where adequate isolation and cycles are available.

For commodity clusters, it may be easier to provide containerized reproductions of the production HPC environment, as there are not as many licensing issues involved. However, since most HPC sites are still administered very manually, HPC administrators will need to lean into a culture of automation. If the site can provision the production environment automatically, they can reliably provide a container with *exactly* the same software that the main HPC site runs.

The idea of building containers based on production HPC environments is not new; one of this paper's authors proposed it with his colleagues to NSF's XD solicitation [5] in 2008. The proposal ultimately became part of NSF's XSEDE environment, which operated between 2011 and 2022, but the containerization component did not make it into the final, funded project.

More recently, NASA Ames [6] has successfully provisioned cloud-bursting capabilities allow users to build, test, and run codes in small allocations in cloud environments before running them unmodified on the production Pleiades HPC cluster. They leverage portable container workflows and abstract differences between cloud and onsite resources through the MPI interface. This pioneering effort to create "reproducible" HPC infrastructure still has security limitations. Even in NASA's environment, the cloud resources are provisioned in the same logical network as HPC onsite resources. They cannot run untrusted code without risk to onsite data.

### More virtualization and IaaS

The solution that likely makes most sense for HPC CI is to move towards a less trusting, more isolated security model that would allow HPC systems to function more like clouds. Flexible, isolated allocations for either internal or external CI jobs would would eliminate the duplication of effort required by many of the potential approaches mentioned above. Isolated allocations also enable Infrastructure as a Service (IaaS) within the HPC site, which would allow sites to mock entire distributed resource manager environments and services. With this capability, developers could test entire workflow systems in much more realistic scenarios.

We will need to work with vendors to develop and provide HPC environments with network and OS support for isolation. In general, the only integrators currently providing these capabilities widely are clouds, and there are *not* good on-premises solutions. HPC sites will need to either start working with clouds more closely, or push HPC vendors to provide like capabilities for HPC centers. It will likely be a long time before truly "converged" infrastructure becomes widespread.

## CONCLUSION

Continuous integration is indispensable for most software development and maintenance today, including for scientific software. However, CI is difficult to implement in HPC environments for many reasons. Limitations of on-premises infrastructure preclude many of the security isolation techniques used in modern cloud environments. HPC security policies must respect these limitations by restricting the automation needed for responsive CI. Current solutions require duplicated effort, either in provisioning dedicated resources for CI, or by duplicating deployment effort with containerized environments. The most promising solution is to move toward more automated, secure, flexible infrastructure, which will be neither quick nor easy to implement with the restrictions of today's HPC environment.

## ACKNOWLEDGMENT

The genesis of this article was presentations by the authors in a panel on Build, Integration and Testing for Sustainable Scientific Computing Software, chaired by Keita Teranishi and Roscoe A. Bartlett, at the 2022 SIAM Conference on Parallel Processing for Scientific Computing. The authors thank the chairs for organizing the panel, inviting us, and creating the opportunity for discussion. DSK also thanks Ben Clifford and other members of the Parsl and funcX teams for some of the ideas here.

## REFERENCES

1. H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon, "Top500 Supercomputer Sites," Online. https://top500.org, 2009.
2. T. Gamblin, M. P. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and W. S. Futral, "The Spack Package Manager: Bringing order to HPC software chaos," in *Supercomputing 2015 (SC'15)*, Austin, Texas, November 15-20 2015, lLNL-CONF-669890.
3. K. Hoste, J. Timmerman, A. Georges, and S. De Weirdt, "EasyBuild: Building Software with Ease," in *High Performance Computing, Networking, Storage and Analysis, Proceedings*. IEEE, 2012, pp. 572–582. [Online]. Available: http://dx.doi.org/10.1109/SC.Companion.2012.81
4. P. Bryant, "Jacamar CI," Online: https://ecp-ci.gitlab.io/, https://gitlab.com/ecp-ci/jacamar-ci, 2019-2022.
5. National Science Foundation, "TeraGrid Phase III: eXtreme Digital Resources for Science and Engineering (XD)," https://www.nsf.gov/pubs/2008/nsf08571/nsf08571.htm, 2008.
6. Y. Chang, S. Heistand, R. Hood, and H. Jin, "Feasibility of running singularity containers with hybrid mpi on nasa high-end computing resources," in *2021 3rd International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. Los Alamitos, CA, USA: IEEE Computer Society, nov 2021, pp. 17–28. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CANOPIEHPC54579.2021.00007